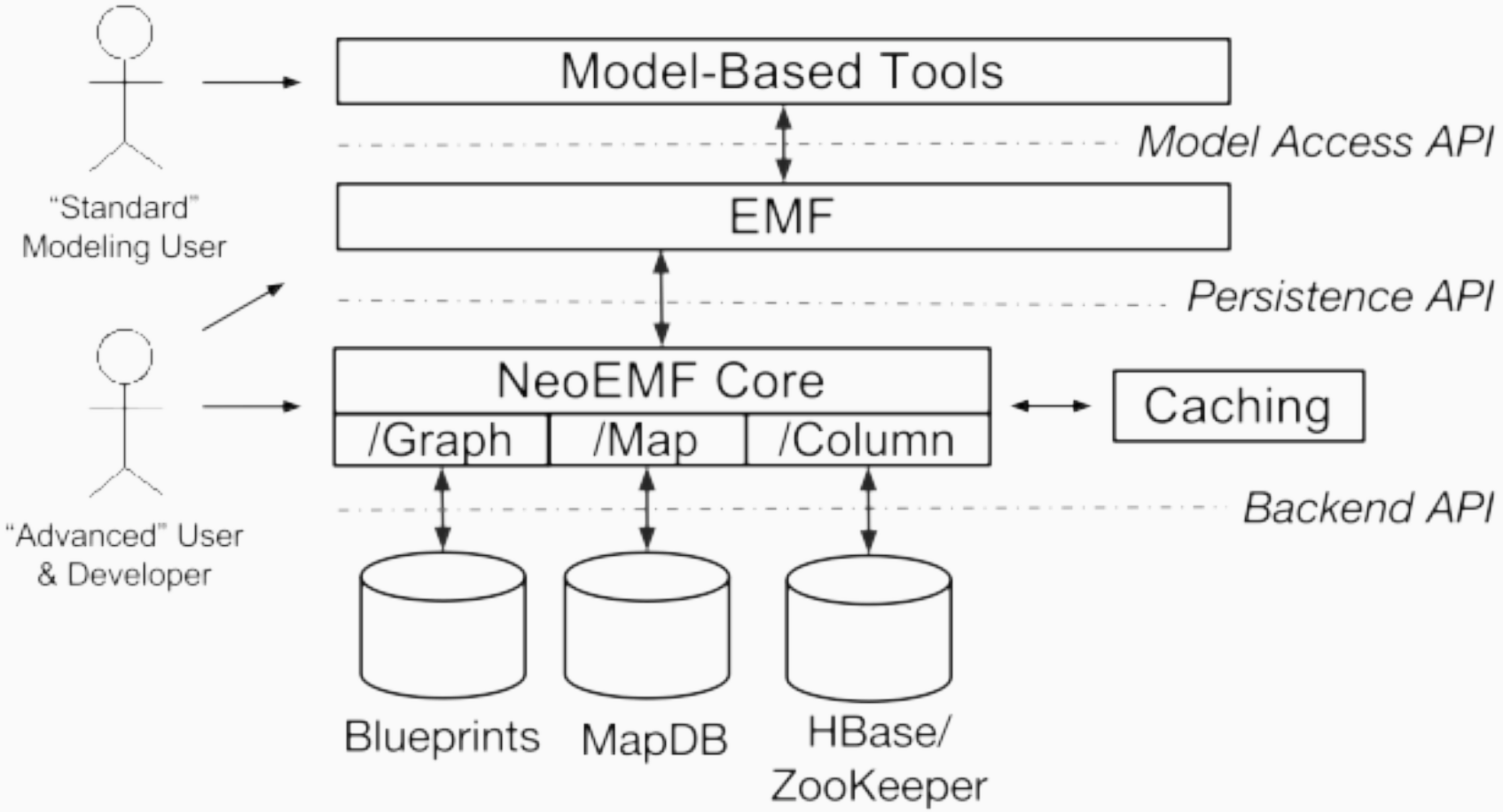
Taming Large Models with Hawk and NeoEMF

A. García-Domínguez, D. S. Kolovos, K. Barmpis, G. Daniel, G. Sunyé

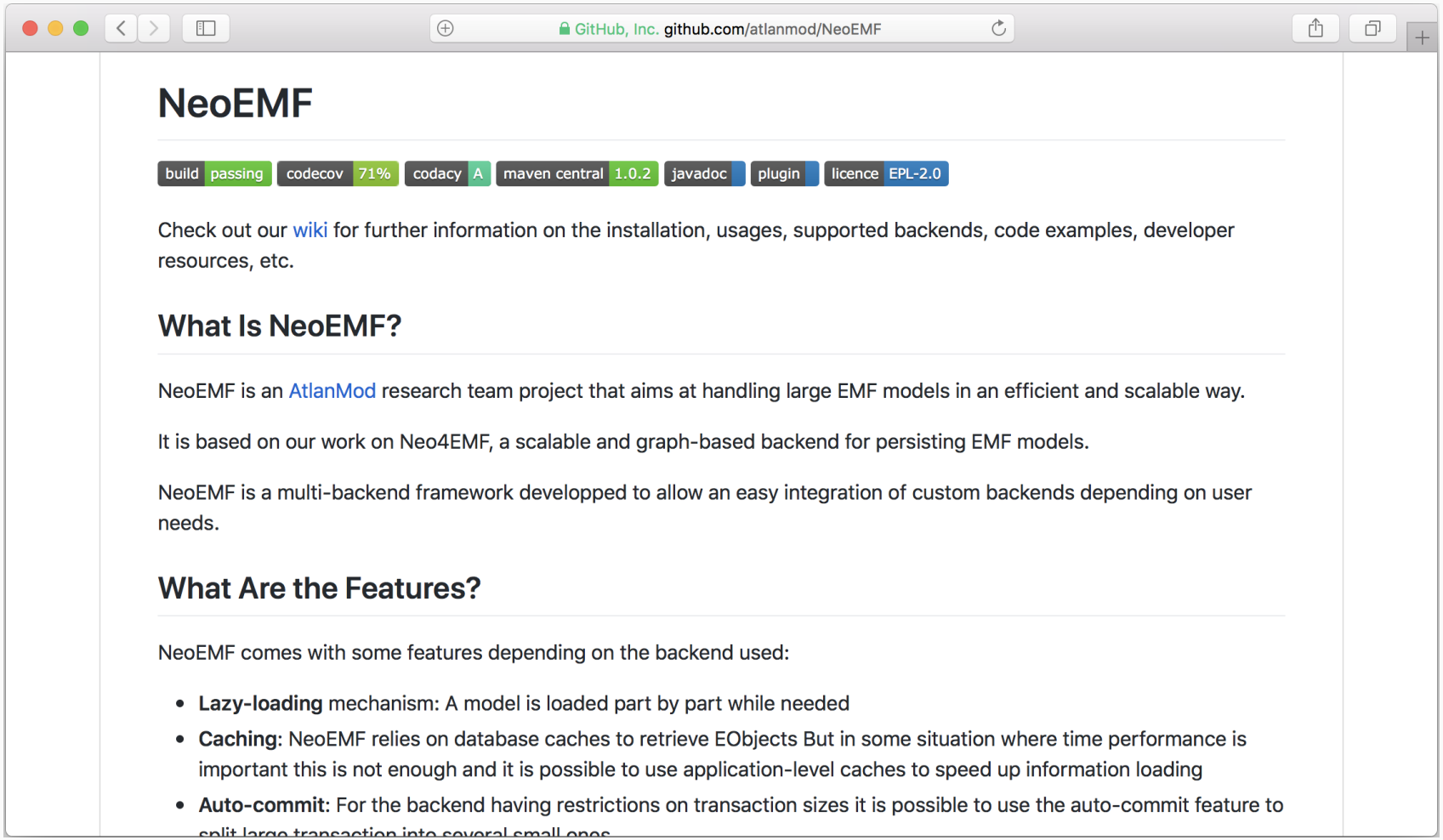
MoDELS'2018, 14–19 October 2018

NeoEMF

NeoEMF: Architecture



NeoEMF: project website



- `https://github.com/atlanmod/NeoEMF`
- Open source project under the Eclipse Public License 2.0

NeoEMF Datastores [DSB+16]

- NeoEMF/Graph
  - Efficient model traversal using rich query language
  - Mogwaï framework (OCL to Gremlin translation)
- NeoEMF/Map
  - Fast access to atomic operations
  - Designed for EMF-API calls
- NeoEMF/Column
  - Transparent model distribution
  - Concurrent read/write
  - Distributed model transformations (ATL-MR)

NeoEMF Key Features

- Lazy-loading
- Compliant with EMF API
- Easy to integrate in existing applications
- EMF-Compatible code generation
- Advanced caching (+ prefetching) strategies
- Efficient XMI importer

Initialise a New Resource

1. Create a new URI to locate a file-based resource.
2. Create the resource.

```
URI uri = MyUriBuilder.builder().fromFile(new File("<db_path>"));

ResourceSet resourceSet = new ResourceSetImpl();
Resource resource = resourceSet.createResource(uri);
```

Save/Load Resource

1. Create a new configuration builder.
2. Save and unload resource.

```
ImmutableConfig config = MyConfig.newConfig()
.autoSave(50000)
.log()
.withOption("key", "value");

resource.load(config.asMap());

// Do something on the resource

resource.save(config.asMap());
resource.unload();
```

Modify existing resource

1. Load resource.
2. Modify contents.
3. Save resource.

```
ImmutableConfig config = MyConfig.newConfig()
.cacheContainers()
.cacheMetaclasses();

URI uri = MyUriBuilder.builder().fromFile(new File("db_path"));

Resource resource = new ResourceSetImpl().createResource(uri);
resource.load(config.asMap());

MyClass myClass = (MyClass) resource.getContents().get(0);
myClass.setName("NewName");

resource.save(config.asMap());
resource.unload();
```

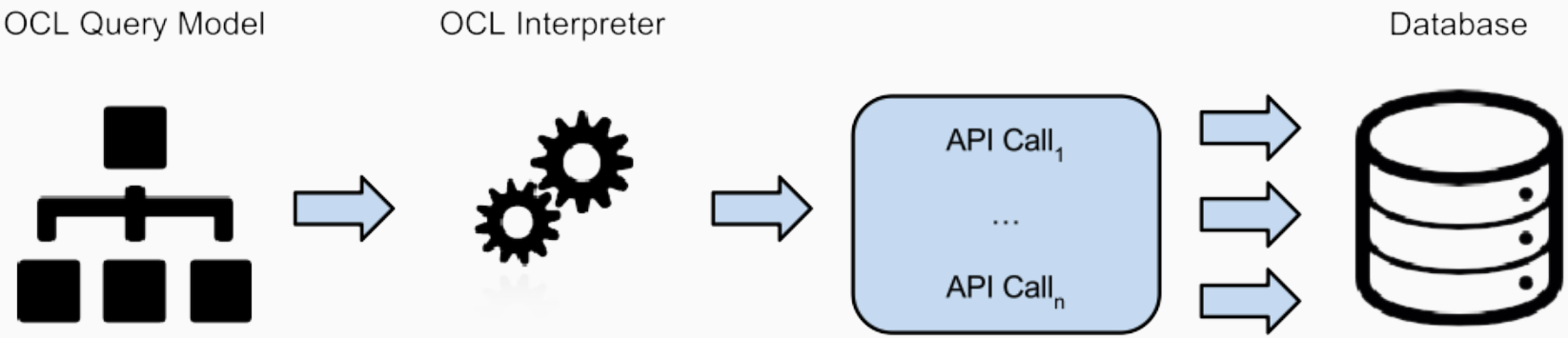
Demo time!

Let's import a Java model, save it in Neo4j and query the database.

Discussion

- Graph databases outperform relational ones for several navigation steps queries.
- But model loading operations only use 2 or 3 steps queries.
- If the use of the EMF API is the only concern, then a relational (or column) database storing BLOBs are a better solution.
- Impossible to compare with bigger models.

Discussion



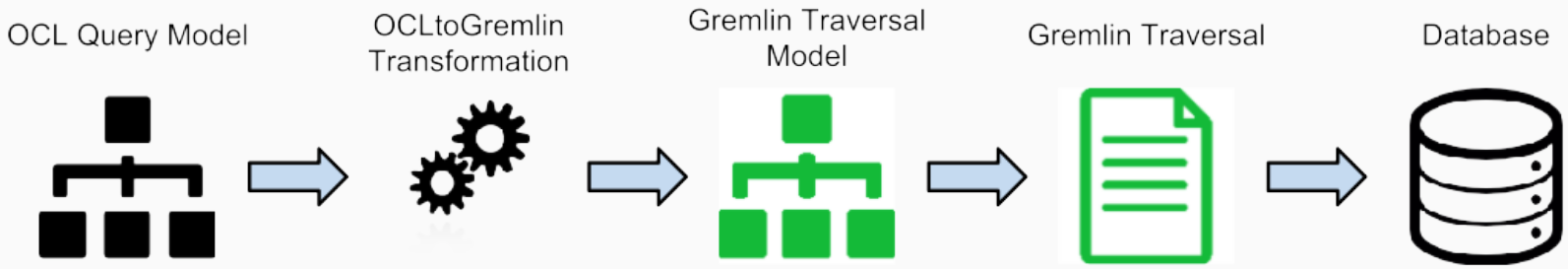
- Low-level model handling APIs
- Fragmented queries on the database
- Many intermediate objects

Mogwaï

Motivation

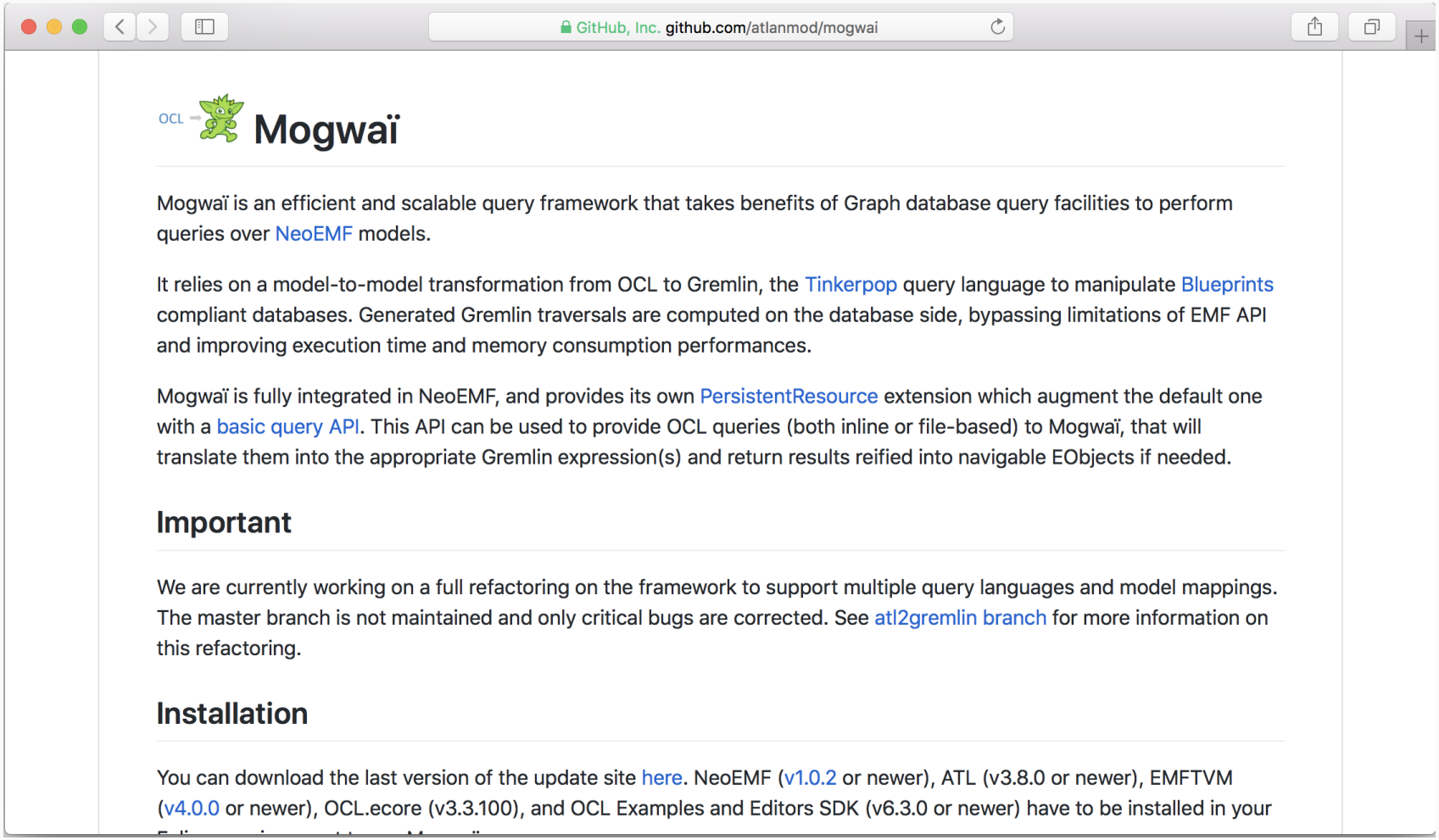
- Why don't we query directly the database?
- Manually writing database-level queries is hard
- Need to learn a new query language
- Database expertise vs. Modeling expertise
- Unknown model representation
- Solution: generate them!

Mogwaï: Architecture [DSC16]



- Generate graph database queries from OCL expressions
- Bypass modelling framework API
- Single execution of the query

Mogwaï: project website



- `https://github.com/atlanmod/mogwai`
- Open source project under the Eclipse Public License 2.0

Queries are expressed in OCL

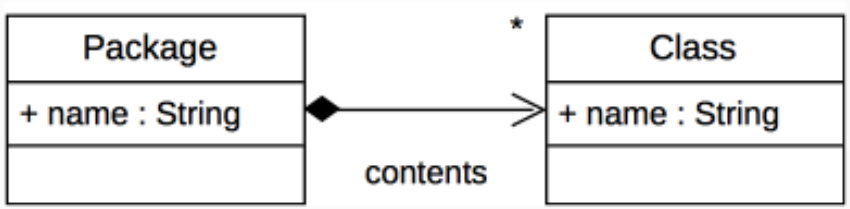


Figure 1: Simple Model

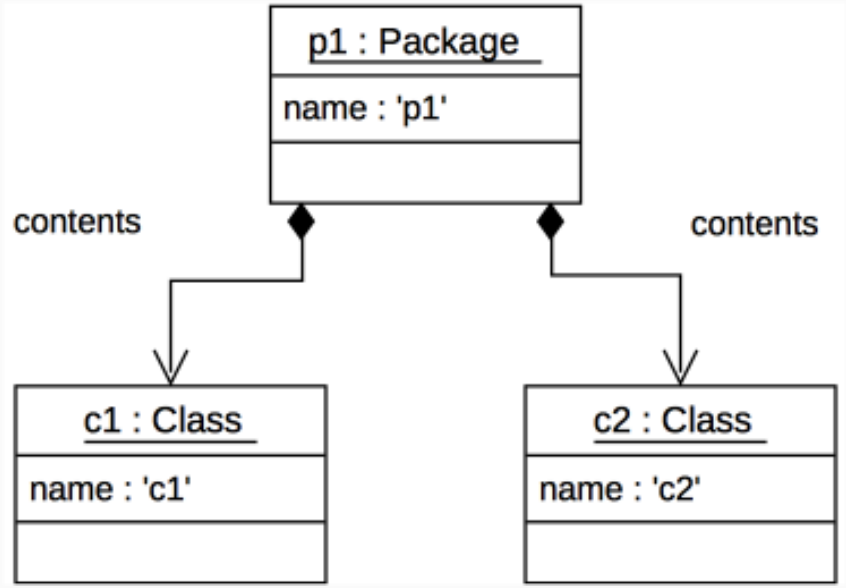


Figure 2: Model Instances

```
        Package.allInstances() --
returns p1
        p1.contents    --returns [c1,c2]
        p1.contents→select(e | e.name = 'c1'
          --returns c1
```

Model Persistence

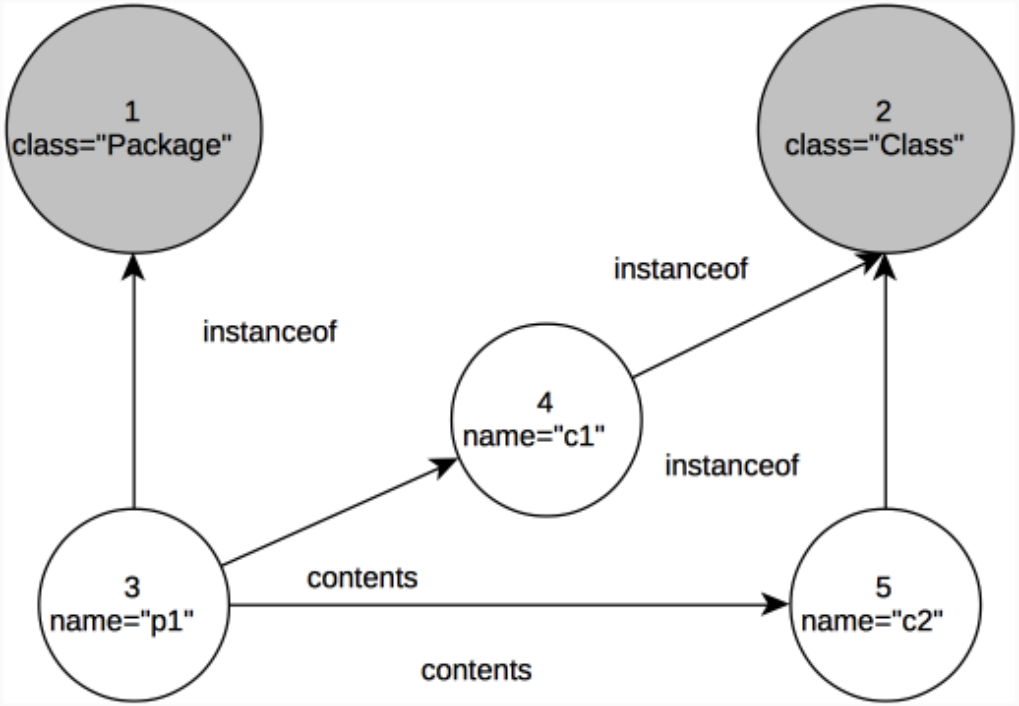
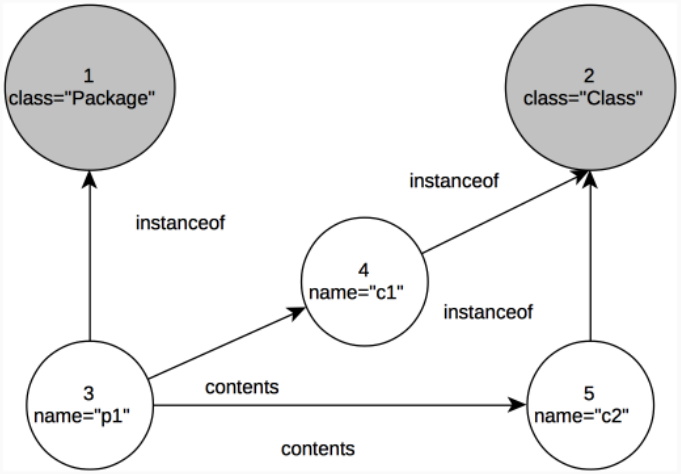


Figure 3: Model Instances Stored in Neo4j

Database Queries are expressed in Gremlins

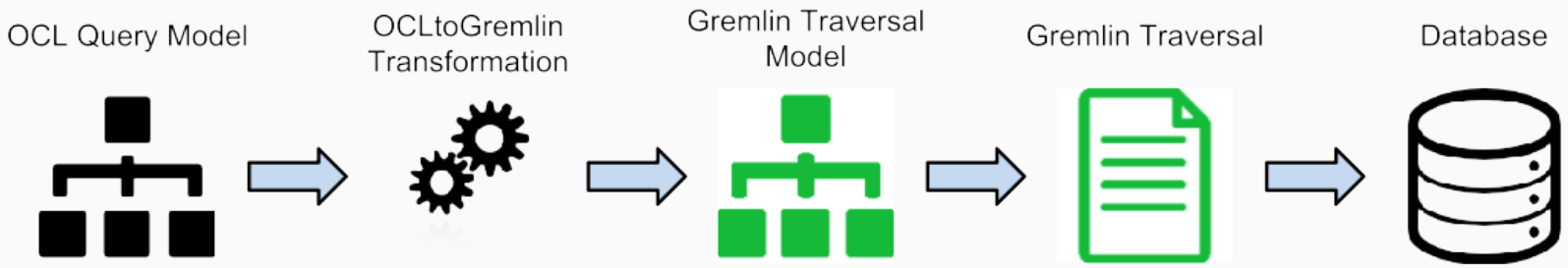
- Graph traversal DSL
- Composed of processing steps
- Generic query language for graph databases



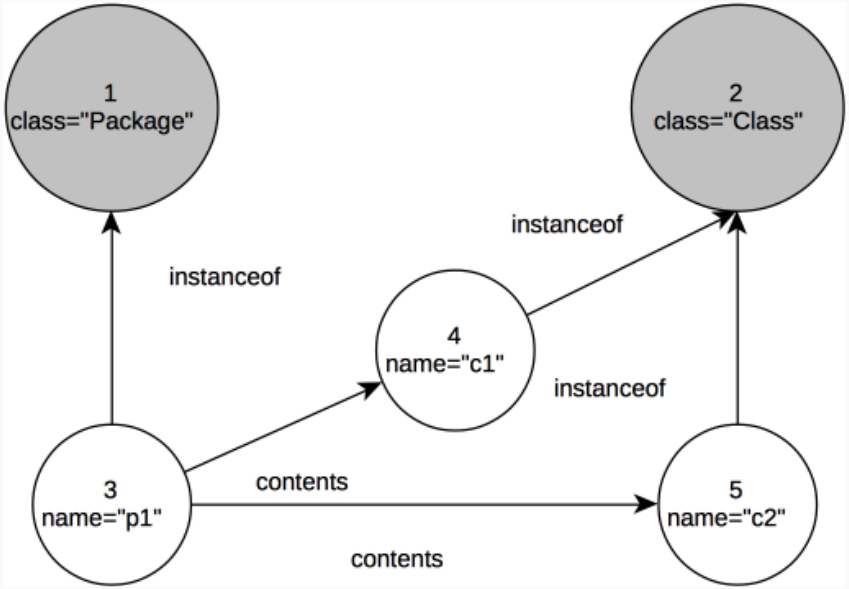
g.**idx**(''metaclasses'')[[name:''Package'']
.**inE**(''instanceOf'').**outV** *// v(1)*
g.**v**(3).**outE**(''contents'').**inV** *// [v(4),v*
g.**v**(3).**outE**(''contents'').**inV**
.**filter**{it.name = ''c1''} *// v(4)*

OCL Queries into Gremlin Traversals Translation

- Map OCL expressions to Gremlin steps
- Merge created steps into a (several) traversal(s)]

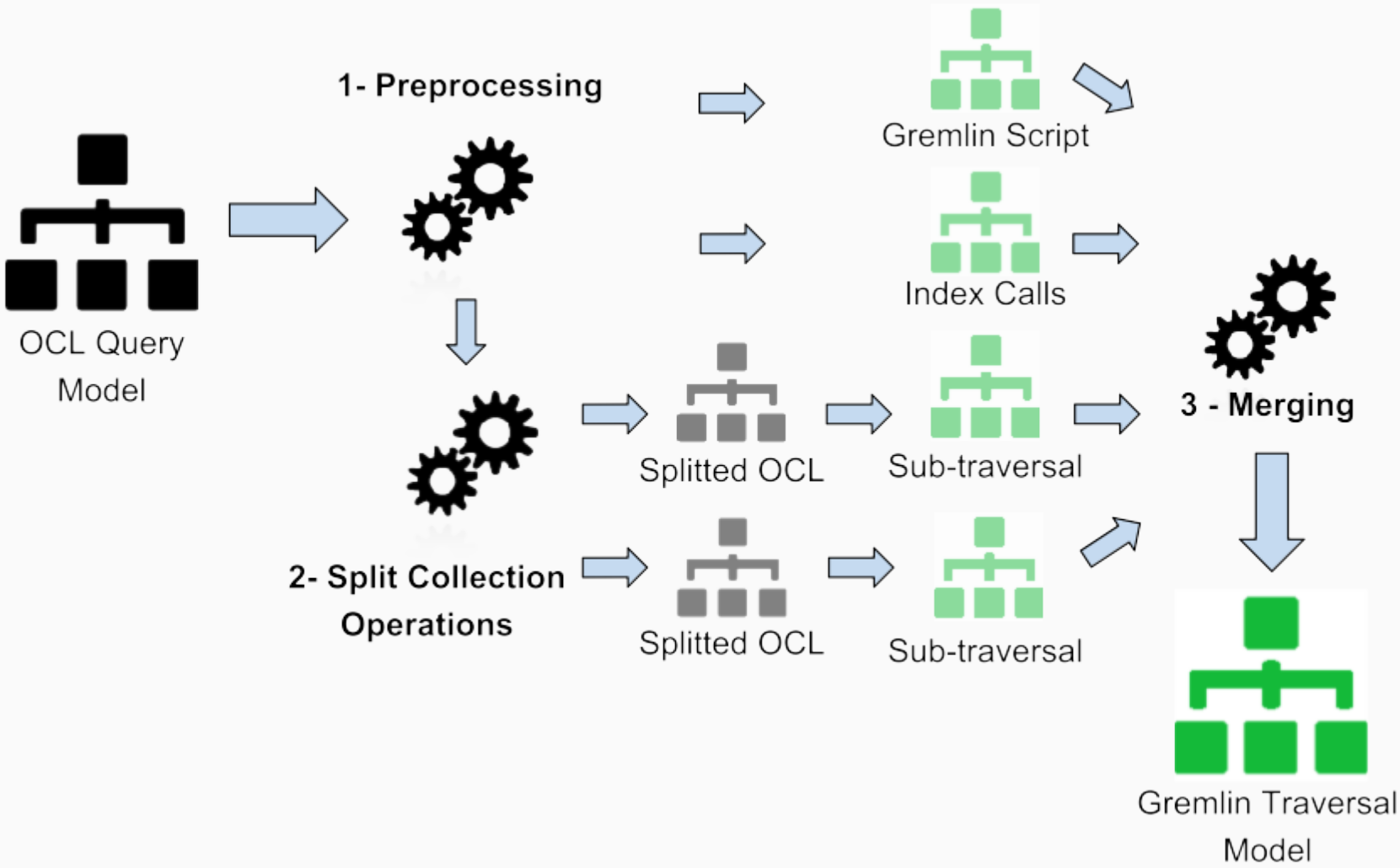


OCL Expressions to Gremlin Steps Mapping

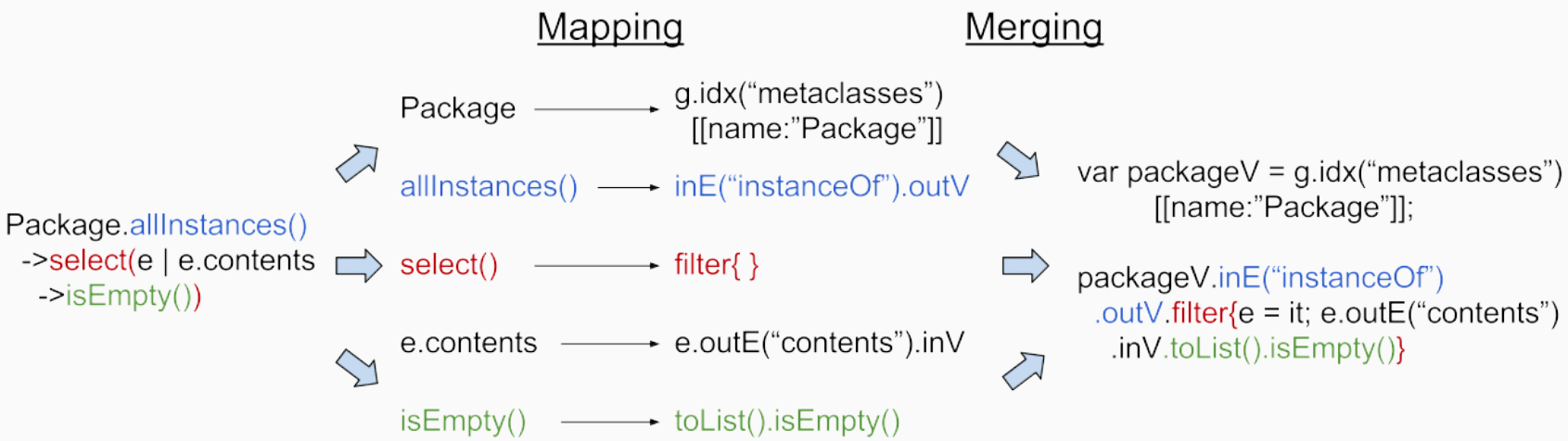


OCL Expression		Gremlin Steps
Type	→	g.idx(“metaclasses”)[[name:”Type”]]
AllInstances()	→	inE(“instanceOf”).outV
collect(att)	→	att
collect(ref)	→	outE(“ref”).inV
select(cond)	→	filter{cond}
oclIsTypeOf(T)	→	outE(“instanceOf”).inV .transform{it.next() == T}

Merge created steps into a traversal

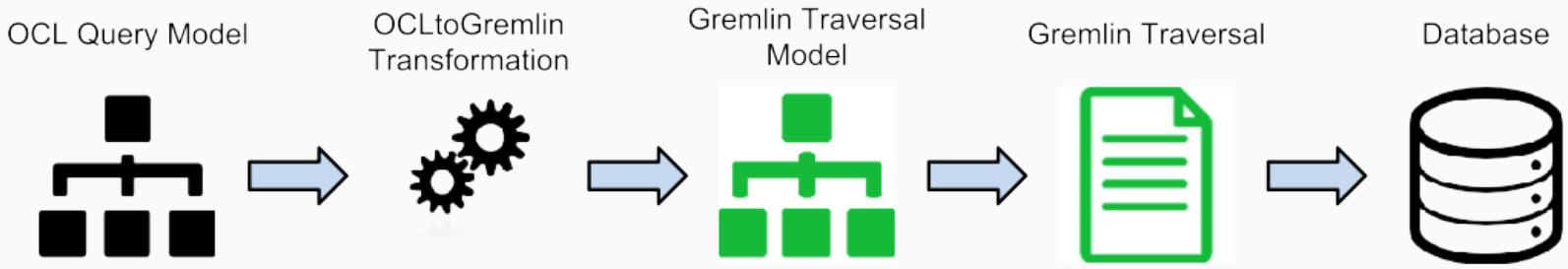


OCL Transformation Example



Query generation and execution

- Delegates query computation to the database
- Returns graph elements to the persistence layer

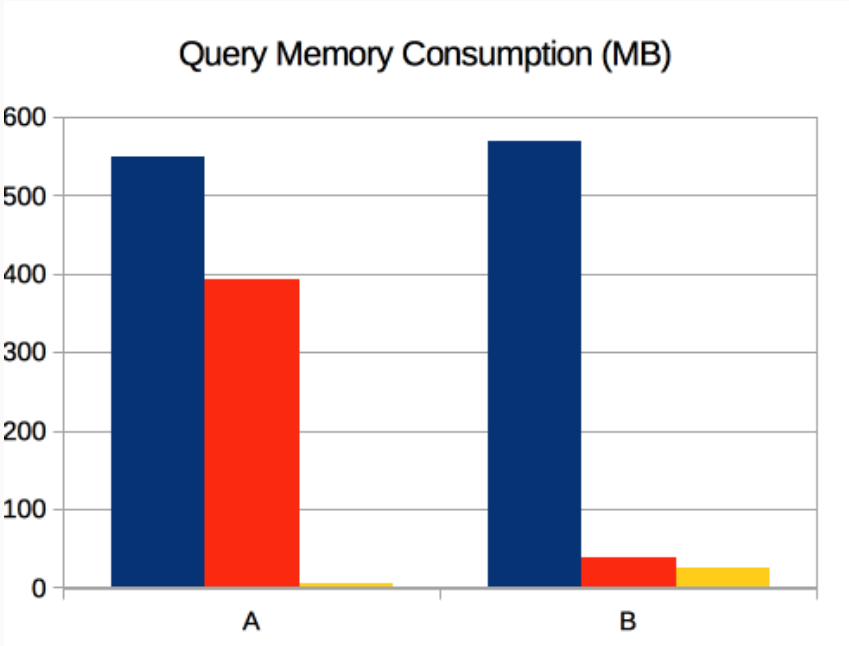


Demo time!

Let's query a Java model and find singletons.

Benchmark Results

- Model containing 2 million elements
- Up to 20 times faster than other query approaches
- Consume up to 75 times less memory



Conclusion

Model Persistence Frameworks

- Not designed to compute model queries efficiently
- Writing manually database-level queries is hard

Mogwaï

- Translates OCL queries into Gremlin traversals
- Positive results
- Not adapted to small models
- Needs to be integrated

References i

Gwendal Daniel, Gerson Sunyé, Amine Benelallam, Massimo Tisi, Yoann Vernageau, Abel Gómez, and Jordi Cabot.
NeoEMF: a multi-database model persistence framework for very large models.
In Juan de Lara, Peter J. Clarke, and Mehrdad Sabetzadeh, editors, *Proceedings of the MoDELS 2016 Demo and Poster Sessions, Saint-Malo, France, October 2-7, 2016.*, volume 1725 of *CEUR Workshop Proceedings*, pages 1–7. CEUR-WS.org, 2016.

Gwendal Daniel, Gerson Sunyé, and Jordi Cabot.
Mogwaï: A framework to handle complex queries on large models.
In *Tenth IEEE International Conference on Research Challenges in Information Science, RCIS 2016, Grenoble, France, June 1-3, 2016*, pages 1–12. IEEE, 2016.